4

A TRIDENT SCHOLAR PROJECT REPORT

NO.166

Development of a Phenomenological Fitting Procedure for the Fast Microcomputer

6

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER U.S.N.A. - TSPR; 166 (1990)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DEVELOPMENT OF A PHENOMENOLOGICAL FITTING PROCEDURE FOR THE FAST MICROCOMPUTER.		5. TYPE OF REPORT & PERIOD COVERED Final 1989/90
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David F. Clipsham		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS United States Naval Academy, Annapolis.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS United States Naval Academy, Annapolis.		12. REPORT DATE 22 May 1990
		13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release; its distribution is UNLIMITED.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Accepted by the U.S. Trident Scholar Committee.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Curve fitting - Computer programs
Phenomenological theory (Physics)
Computer simulation
Turbo Pascal (Computer program)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Currently available fitting techniques for both linear and nonlinear models are slow, require expensive equipment that is not widely available, and shield the modeler from an in-depth understanding of the fitting procedure. New high speed microcomputer technology has opened the door for the development of microcomputer based fitting techniques that eliminate some of these shortcomings. This paper begins by discussing fitting in

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

general with an emphasis on the distinction between the linear fitting problem and the non-linear fitting problem. The suitability of existing fitting techniques to handle these fitting problems is analyzed, and the strengths and weaknesses of each technique are noted. A case is made to justify the development of a microcomputer based phenomenological fitting procedure, and the technique is developed. The various stages in the development of the technique are described including the arrangement of a user-friendly display, the choice and testing of the cybernetic interface, and the writing of the computer code in Turbo Pascal 5.5. The technique is validated by fitting a number of real world data sets, and two of these validation tests are discussed in detail. Finally, speculation is made about the future of the phenomenological fitting technique. The phenomenological fitting algorithm developed in the paper is included on disk with a brief user's manual.

U.S.N.A. - Trident Scholar project report; no. 166 (1990)

Development of a Phenomenological Fitting Procedure for the Fast Microcomputer

A Trident Scholar Project Report

by

Midshipman David F. Clipsham, Class of 1990

U. S. Naval Academy

Annapolis, Maryland

[illegible]

Associate Professor John P. Ertel
Physics Department

Accepted for Trident Scholar Committee

[illegible]

Chairperson

22 May 1990

Date

USNA-1531-2

ABSTRACT

Currently available fitting techniques for both linear and non-linear models are slow, require expensive equipment that is not widely available, and shield the modeler from an in-depth understanding of the fitting procedure. New high speed microcomputer technology has opened the door for the development of microcomputer based fitting techniques that eliminate some of these shortcomings.

This paper begins by discussing fitting in general with an emphasis on the distinction between the linear fitting problem and the non-linear fitting problem. The suitability of existing fitting techniques to handle these fitting problems is analyzed, and the strengths and weaknesses of each technique are noted. A case is made to justify the development of a microcomputer based phenomenological fitting procedure, and the technique is developed.

The various stages in the development of the technique are described including the arrangement of a user-friendly display, the choice and testing of the cybernetic interface, and the writing of the computer code in Turbo Pascal 5.5. The technique is validated by fitting a number of real world data sets, and two of these validation tests are discussed in detail.

Finally, speculation is made about the future of the phenomenological fitting technique. The phenomenological fitting algorithm developed in the paper is included on disk with a brief user's manual.

ACKNOWLEDGMENTS

I would like to take the opportunity to thank the many people without whom this project could not have been successfully completed. For their patience in dealing with my continuous flow of phone calls and my constant presence in their office, I extend my thanks to Marge Bem and Martha Blair. The arduous task of coding *FlyFitter* was made infinitely more tolerable by the sage advice of Dr. Allan Herr, and by the helpful hints offered by LCDR William "Buzz" Borries. The function parser, originally written by Assistant Professor Frank Pittelli, and the library of math functions that make model entry so easy in *FlyFitter* were generously made available to me by Associate Professor James Buchanan and Professor Howard Penn who also made themselves available to answer my many programming questions. For helping me with the details of preparing for the Trident Dinner, my gratitude goes to Professor Larry Tankersley, Rachel Heberle, and Dave Rector. Testing the program could not have been done nearly as effectively without the data and explanations provided by Associate Professor Murray Korman. My special thanks go out to my Trident Advisor, Associate Professor John P. Ertel whose eccentric genius infused me with interest in the project and whose encouragement and ever-positive outlook kept me going through the rough spots. Finally, this research project would not have been possible without the generous support of the Naval Academy Research Council and the Trident Scholar Committee.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

It has been the ultimate goal of science since its fledgling years to find a model of the universe that would enable mankind to predict the future with absolute certainty. Needless to say, this goal has not yet been achieved. However, many less general models have been developed to help predict the behavior of millions of common phenomena. For example, if a net force F is applied to an object of mass m in a frictionless environment, Newton's Second Law predicts that the acceleration a of the object will be $a=F/m$. The equation $a=F/m$ is a mathematical model used to describe the behavior of a mass—net-force system in a frictionless environment. Given any two of the quantities $a, m,$ or F, the third quantity can be directly calculated. Alternatively, given that one of the quantities, m, is constant, a plot may be constructed of a vs. F. The problem, then, is to determine a particular value for m. It may be seen that mathematical modeling involves two parts: formulation of a mathematical relationship between the dependent and independent variables, and determination of the values of the parameters in the relationship.

If, as a more complex example, an object of mass m is attached to a spring of spring-constant k, submerged in a viscous fluid, displaced a distance x_0 from its equilibrium position, and allowed to oscillate, it is not obvious what the displacement of the mass from equilibrium will be at some time t in the future. The goal of a modeler in a situation like this might be to develop a model that incorporates all possible variable quantities and that yields a value for the displacement x for any valid value of time. Such a model might look something like

$$x(t) = x_0 e^{-(t/\tau)} \cos(\omega t) \quad ,$$

where the damping coefficient for the system is τ and the frequency of oscillation for the system is ω. Once the model has been developed and validated, it can be put to use in a number of different ways. The most obvious way is that given values for x_0, τ, and ω, the model could be used to predict a displacement for the mass at any given time. In the real world, however, quantities like the damping coefficient and the frequency of oscillation are often not directly measurable. Instead, easily measured quantities like time and displacement must be used to determine values for those quantities not easily measured. If the apparatus described above were set up in a laboratory and data points were taken consisting of matching time and displacement pairs, the data could be graphed and, presumably, values could be determined for x_0, τ, and ω. Clearly there is no way to uniquely determine values for each of the unknown quantities in the model because it is not possible to eliminate the other unknown quantities from the solution. Substitution into the model of a single set of values for time and displacement taken in the lab would result in an infinite number of possible combinations of values for x_0, τ, and ω that satisfy the model. Additionally, uncertainties in the measurements of time and displacement taken in the lab introduce uncertainty into the values that are determined for x_0, τ, and ω [TAY82]. The problem, then, is to determine the values for x_0, τ, and ω that best satisfy all of the data points taken in the laboratory simultaneously. This problem and all similar problems fall under the general category of curve fitting. The general term, parameter, is used for the unknown quantities whose values are being sought.

Goodness of Fit

Before beginning to fit a model to a set of data, it is necessary to determine under what conditions the fit will be satisfactory. In other words, goodness of fit criteria must be established. The quantity most commonly used in determining goodness of fit is chi-square (χ^2) which is the sum of the squared deviations for the ith measurement, each weighted inversely as the square of the uncertainty, σ_i, in the measurement

$$\chi^2 = \sum_{i=1}^{NP} \left[\frac{1}{\sigma_i^2} (y_i - y_t(x_i))^2 \right] .$$

Defining the number of degrees of freedom, ν, for the fit as one less than the difference between the number of points, NP, and the number of terms, NT, or parameters in the model, $\nu \equiv NP - NT - 1$, the associated measure of the goodness of fit called reduced chi-square (χ_ν^2) is obtained. χ_ν^2 may be defined as the goodness of fit per degree of freedom

$$\chi_\nu^2 \equiv \frac{\chi^2}{\nu} .$$

If χ^2 is plotted in parameter space for a model with two parameters (Figure 1), the conditions under which a best-fit would occur are more easily visualized. The surface defined by χ^2 plotted against the parameters in the model is generally referred to as a hyper-surface because if more than two parameters exist in the model, the plot is not a true surface and the visualization becomes somewhat more difficult. The location of a minimum in the χ^2 hyper-surface defines the parameter values which yield the best fit.

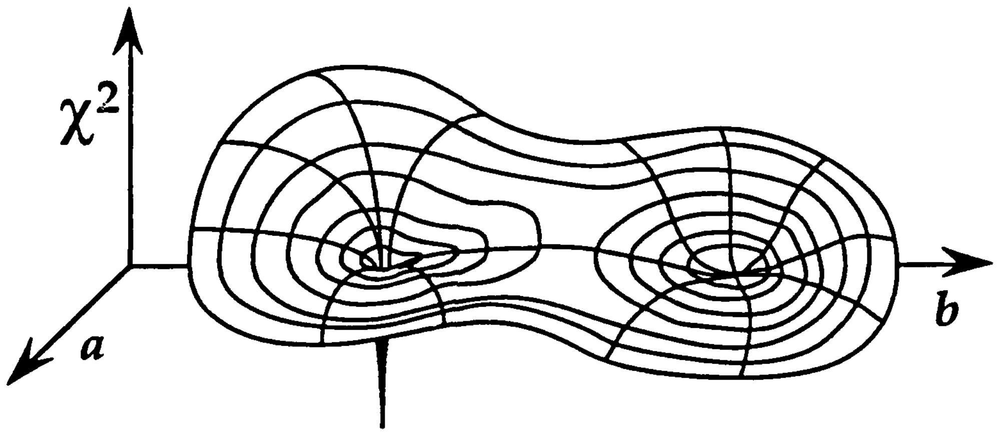


Figure 1: Chi-Square (χ^2) Hyper-Surface

The successful determination of the parameter values in a modelling situation like the one described above depends very heavily on the choice of the model. For the data taken in the lab from the damped mass-spring system discussed above, a damped sinusoidal model was chosen. In a situation where the behavior of the system is not well understood, a much less appropriate model might be inadvertently chosen. For example, had $a + bx$ been chosen as the model instead of the damped sinusoid, values for a and b would still have been determined in the same way, but the resulting model would have been far less successful in predicting the behavior of the system.

All mathematical models fall into one of two categories: linear or non-linear. Some models are explicitly non-linear, but can be linearized through algebraic manipulation. These models will be called linearizable models, and they fall under the category of linear models for the purpose of fitting them to sets of data.

Linear & Linearizable Models

Models that are truly linear are those in which the parameters actually appear linearly in the model. The power series defining the theoretical values, $y_t(x_i)$, as

$$y_t(x_i) = a + b x_i + c x_i^2 + d x_i^3 + \cdots$$

is a linear model because all of the parameters $a, b, c, \ldots$, appear linearly. The linear model may be expressed generally as:

$$y_t(x_i) = \sum_{n=0}^{NT-1} a_n \mathcal{F}_n(x_i) \quad .$$

A linearizable model is a model in which the parameters do not explicitly appear linearly, but which has a well defined and directly calculable inverse. This definition may be expanded for the purposes of this discussion to include the requirement that the inverse be <u>easily</u> calculable because if it is not, linearizing the model yields no savings in time or effort on the part of the modeler. The model

$$y_t(x_i) = a\,x_i^b\,e^{c\,x_i}$$

is not explicitly linear in x and y, but, if x and a are assumed positive, it can be linearized by taking the natural log of both sides to yield

$$\ln[y_t(x_i)] = \ln a + b\ln x_i + c\,x_i \quad ,$$

which is linear in x, $\ln a$, $\ln$x, and $\ln$y [DAN80].

The χ^2 hyper-surface for a linear or linearized model and any set of data has a single well defined minimum in it that can be easily found with a microcomputer or, in many cases, with a programmable hand-held calculator. The location of the minimum defines a unique set of best fit parameters for the model and data.

Non-Linear Models

Truly non-linear models are functions that have no directly calculable inverse. For example,

$$y_t(x_i) = a\,x_i + b\,e^{c\,x_i}$$

is not linear in x and y, and it cannot be linearized because it has no directly calculable inverse.

A typical χ^2 hyper-surface for a set of data and a non-linear model has many relative minima rather than the single absolute minimum seen with a linear or linearized model. As a result, many possible combinations of the model parameters will result in a relative minimum in χ^2. This greatly complicates the job of the modeler because it must then be decided which minimum and corresponding set of model parameters is being sought. Even once it is decided which minimum is being sought, it is no easy matter to find it. A number of techniques exist to help modelers search parameter space for relative minima in χ^2.

EXISTING FITTING TECHNIQUES

Curve fitting is generally a very computationally intensive task. Most implementations of curve fitting techniques, therefore, require a computer or at least a hand-held calculator. There are a number of different techniques that are used to determine the best fit parameterization of a mathematical model to a set of data, but most of them fall under one of two main categories: linear regression and non-linear regression.

All linear and linearized models may be fit to a set of data using linear regression techniques. In the case of very simple linear models, a hand-held calculator can easily determine the parameter values which best fit a model to a set of data. The model $a + bx$ is a very frequently used model, and formulae have been developed that yield values for a and b given a set of data pairs. The fitting is simple because only one minimum exists in the χ^2 hyper-surface. As a result, the best fit values for a and b are unique. For a more complex model like a fourth or fifth order polynomial, computer programs must be used. The process for determining the coefficients of such a polynomial being fit to a data set is now iterative and computationally intensive, but it will still yield a unique and easily determined set of best-fit parameters.

An explicitly non-linear but linearizable model must be linearized before linear regression techniques can be applied to it. The process of linearizing the model is called Functional Linearization (FL). Once the model has been linearized, linear regression techniques will again yield a unique and easily determined best fit parameterization.

Non-linear regression techniques must be applied to models that are non-linear and cannot easily be linearized through FL. Grid/Gradient Search (GGS) and parabolic interpolation of relative minima are two well known techniques of non-linear regression [BEV69]. Less well known are more recently developed techniques like the Eigen-Valued Approach to Modelling (EVM) and finding the L2-Norm Solution of the Hessian Matrix (NL2SOL). A detailed analysis of these techniques is not within the scope of this discussion, but all of them have a number of things in common that are worth considering.

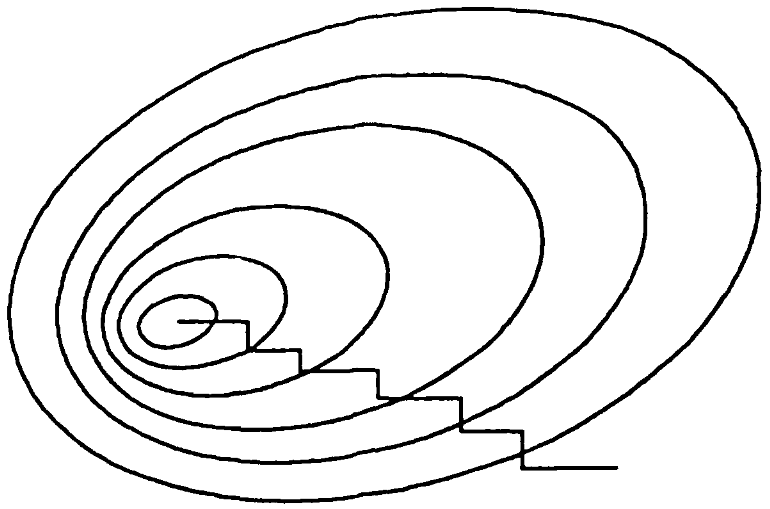


Figure 2: Grid/Gradient Search (GGS)

All of them require the modeler to provide the computer with initial estimates of the values for the model parameters called seeds. They then utilize various techniques to search parameter space around the area of the initial parameter values and attempt to vary the parameters in such a way that they "fall into" the nearest or deepest local minimum in the χ^2 hyper-surface. For example, the GGS technique (Figure 2) solves for the gradient of the hyper-surface at various points in parameter space and follows the gradient directions into local minima. Similarly, the parabolic interpolation method (Figure 3) works by varying a parameter in a direction of decreasing χ^2 until an increase in χ^2 is encountered. It then assumes the shape of the local minimum may be approximated by a parabola and uses the three most recent parameter values to define the parabola and solve for its minimum. The same technique is repeated again for each of the parameters in the model.

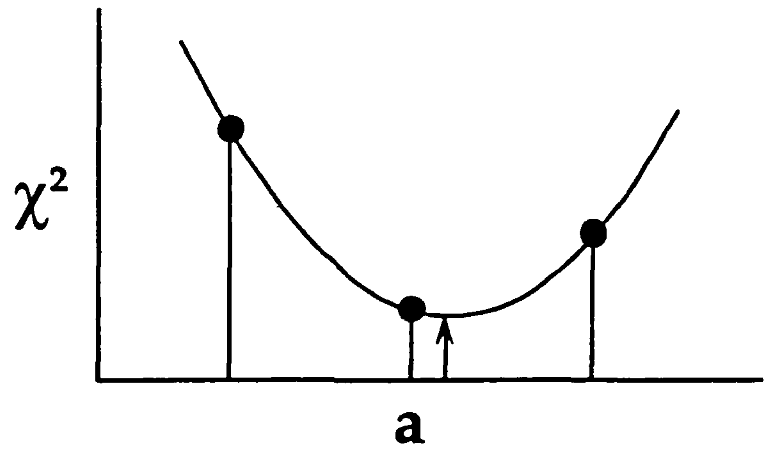


Figure 3: Parabolic Interpolation

Clearly these techniques are very computationally intensive and often must be implemented on large main-frame or mini-computers. Main-frame and mini-computers are usually difficult to find, and computing time on them is scarce and expensive. Even on fast machines, complex fits can require hours or even days of costly computer time. Additionally, since the hyper-surface of a non-linear model typically includes many relative minima, it is easy to see that given the wrong initial parameter seeds, these techniques will find the wrong minima. Since there is usually no way to interact with the computer while it is fitting, the only remedy for this problem is to provide it with new seeds and try again. Unless the modeler has a good idea where the minimum being sought is located and can restrict the variation of the parameters to within pre-specified limits, the computer may well waste expensive time searching in parameter space where the values for one or more parameters are non-physical. Searching parameter space, for example, in an area where a parameter that represents frequency has a negative value is an exercise in futility. Since there is no way to compare the data set and the model visually with conventional non-linear regression techniques, it is not easy to determine the appropriateness of a model being fitted until the fit has been completed and a graph of the data and the model can be compared. It is specifically with these shortcomings in mind that research was conducted into and a program developed to implement the phenomenological fitting procedure.

PHENOMENOLOGICAL FITTING

Phenomenological fitting is interactive curve fitting. It works on the principle that human logic coupled with a computer's number crunching ability is often more effective than brute-force computer techniques at fitting curves to data. More specifically, it relies on the fact that a curve that appears to fit a set of data usually does fit the data. A technique that allows real-time human interaction and participation coupled with a graphical display of the fitting process has the following inherent advantages over conventional fitting techniques:

1. In general, experience strongly suggests that a curve that appears to fit a set of data visually, fits the data statistically.
2. Non-physical parameter situations can be recognized and easily avoided.
3. A feeling is obtained for the sensitivity of the model to variations in individual parameters.
4. A feeling is obtained for the interaction of two or more parameters.
5. The range of applicability for the model is more easily understood.

6. The equipment required is relatively inexpensive and widely available.
8. The technique allows the user to easily obtain parametric seeds for use in more rigorous fitting procedures.

In its pure form, phenomenological fitting also has the following disadvantages:

1. The technique is not statistically rigorous.
2. Current technology restricts the technique to fits involving a single independent variable.

Developing and implementing the phenomenological curve fitting technique was the primary objective of this research project.

Since one of the objectives of the research was to make phenomenological fitting available to a wide range of individuals and to make it inexpensive, it was decided to develop the program on a microcomputer. Turbo Pascal was used as the programming language with some assembly language inserted where speed was critical. The program in its current form is called *FlyFitter* because the process of phenomenological fitting reminds one of "flying" a fit.

In order for a human mind to be able to compare a set of data and a mathematical model quickly and accurately, a plot must be displayed with the model superimposed on the data. The modeler must be able to change the model parameters with precision and ease once it has been decided what parameter modifications are necessary. Finally, once the parameters have been modified, the old model curve must be erased and the new one displayed as quickly as possible so that the fitting can proceed smoothly. These three requirements which can be summarized as visual clarity, ease of use, and speed of operation, were the primary elements considered in developing *FlyFitter*.

The Display

In developing the optimum on-screen display for *FlyFitter*, four visual elements were of primary concern. These four elements were screen composition, display arrangement, color, and resolution. It was determined that certain information has to be available to the modeler during the fitting process if the fitting is to proceed quickly and smoothly. The necessary information includes: the designations and values of each of the parameters, the current value of χ^2, a plot of the model superimposed on the data, and information about how to vary each of the parameters.

Several different arrangements of the screen were tried in an effort to display all of the necessary information in an organized and easily read fashion. Some of these arrangements are shown schematically in Figure 4. Screen (d) was selected for use in the program because it offered the optimal plot-area aspect ratio, and it had more room than the other screens for displaying parameter information. It was also found to be the most natural arrangement of information for scanning the various information blocks during fitting.

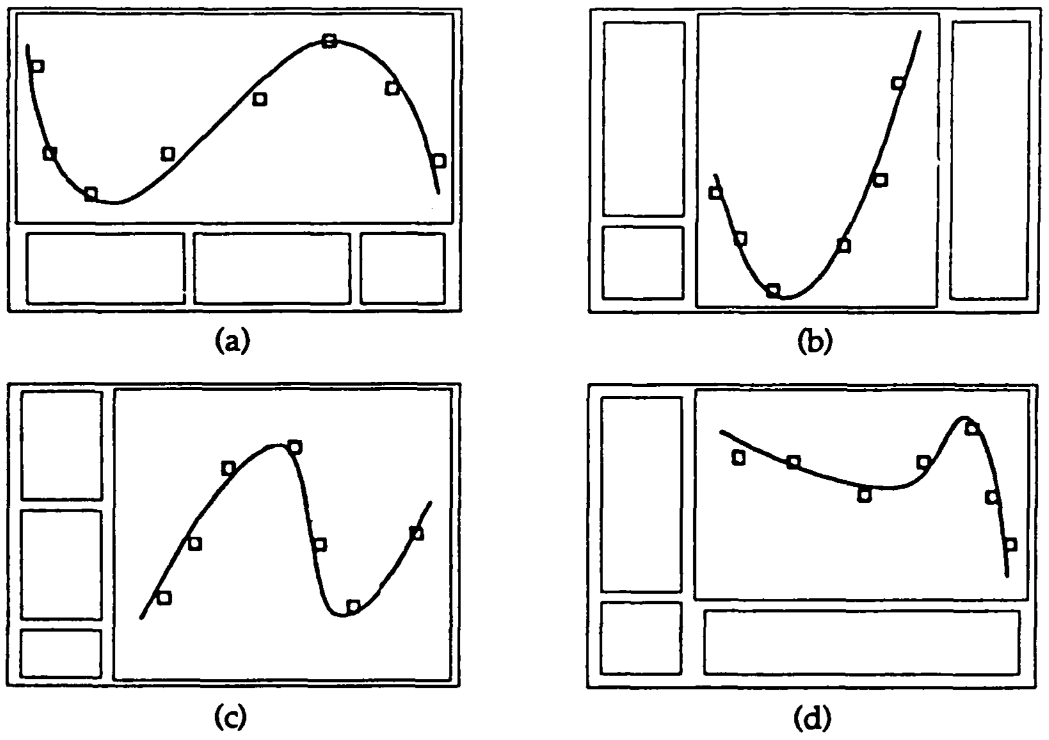


Figure 4: Trial Screen Arrangements

Since computers with color monitors are widely available today, colors were utilized in *FlyFitter* to help clarify the display of information on the screen. Only two colors were used on the fitting screen because it was found that using more than two colors was often distracting to the modeler. Light-gray was chosen as the background color and the color for plotting the data because it isn't harsh or difficult to look at for extended periods of time. Light-green was used to display the model and the values of the model parameters and χ^2 because again it is easy to look at, yet it contrasts well with the light-gray background.

Clearly, for an application like phenomenological fitting that relies heavily on the precise visual display of data, high screen resolutions are desirable. However, since it takes more time to compute and draw high-resolution curves, speed had to be considered in choosing a graphics mode for the fitting screen. Additionally, some new graphics cards are too new to meet the requirement that they be widely available. The Color Graphics Adaptor (CGA) is very widely available, but it does not support high enough resolutions to make phenomenological fitting practical. The much newer Video Graphics Array (VGA) is becoming more widely available, but it is not as well established as the Enhanced Graphics Adaptor (EGA), and it does not offer significant improvements in resolution over the EGA. Furthermore, VGA graphics modes are not supported by older cards like the well established EGA card. The 640 pixel by 350 pixel 16 color EGA graphics mode was selected for the fitting screen in *FlyFitter* because of the EGA card's wide spread availability, the suitability of the resolutions and colors offered, and the fact that newer graphics cards are compatible with the 640 x 350 16 color EGA mode [SUT88].

The Cybernetic Interface

By its nature, phenomenological fitting requires that some means be available for the modeler to easily interact with the computer. Ideally, the means selected offers the most user-friendly interface for the task being performed. Some of the means considered in designing *FlyFitter* were the keyboard, the biaxial joystick, the mouse, the trak-ball, the touch tablet, and the touch sensitive screen.

There are three different ways of varying parameters using these devices. The first way is to designate which parameter is to be varied before variation begins then control the rate of variation of the parameter with the amount of movement of the device being used. If, for example, a modeler using a biaxial joystick wished to increase a model parameter *a* that represents angle of attack, the parameter must first be assigned to an axis on the joystick (Figure 5). Pushing the joystick to the right would then result in continuous increase in the value of *a*, and pushing the joystick to the left would result in continuous decrease in the value of *a*. The rate at which *a* is increased or decreased would be proportional to the angle at which the joystick was deflected. Variation of the parameter would be halted by returning the joystick to the center or neutral position. This method of variation allows the modeler to vary more than one parameter at once. For example, the modeler might assign another model parameter, *b*, representing airfoil thickness, to the vertical axis of the joystick. The number of parameters that can be simultaneously varied is limited only by the number of input devices that may be read by the computer, the speed of the computer, and the ability of the modeler. This method of parameter variation lacks the precision of the other two types, so it requires good hand-eye coordination to be effective, especially if more than one parameter is being varied at once.

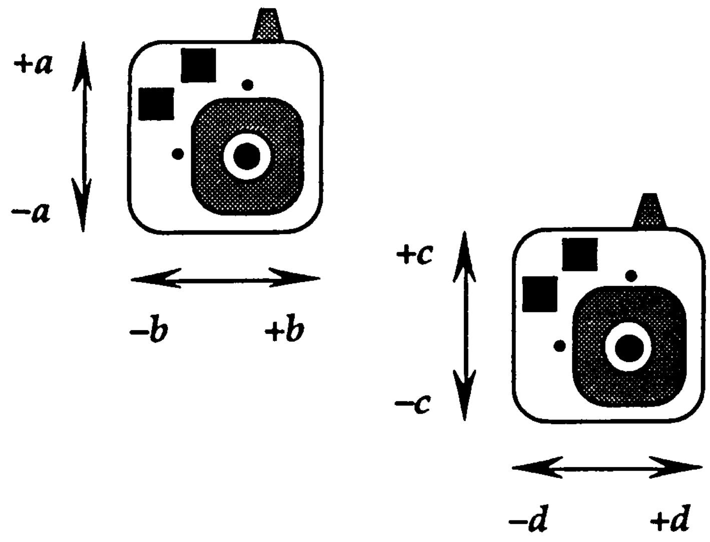


Figure 5: Definition of Joy-Stick Axes

A second way of varying parameters using the devices listed above is similar to the first. As before, the parameter to be varied must be designated in advance. However, instead of using the input device to control the <u>rate</u> of parameter variation, the modeler uses it to control the <u>amount</u> of parameter variation. If, for example, a mouse were being used to vary the angle of attack *a*, some constant of proportionality would be designated by the modeler to define the relationship between the amount of mouse movement and the amount of parameter variation. The modeler could instruct the computer to vary *a* by 2 degrees for every inch the mouse is moved. The angle of attack could then be increased or decreased by precise amounts by moving the mouse to the right and left respectively. Another parameter like airfoil thickness *b*, could be assigned to vertical movement of the mouse allowing simultaneous variation of multiple parameters as with the first method, but with greater precision than before.

A third way of varying parameters is to designate the amount that a parameter is to be varied before variation begins, then control which parameter is varied and for how long using the input device. Suppose, as in the previous example, that a modeler wishes to vary the angle of attack, α, in a model. Before beginning variation of α, the modeler must designate an amount by which α is to change every time a button is pressed on the input device. The input device must then be manipulated so that an on-screen pointer designates α from among the other parameters in the model as the one to be varied. Pressing a button on the input device then varies the parameter by the specified amount. Holding the button down results in continuous variation of the parameter at a rate proportional to the designated variation amount. This third method of variation is the most precise of the three methods, but it offers no practical way to vary more than one parameter at a time.

After considering the merits of each input device and the three methods of parameter variation, it was decided to write two versions of *FlyFitter* with the intent to discard one when it proved significantly inferior to the other. *FlyFitter* was originally envisioned using biaxial joysticks, so one of the versions was written using two biaxial joysticks that varied parameters using the first method described above. The second version was written using a mouse because mice are so widely available, and the third method described above was selected to vary the parameters because it offered the most precision. Since a trak-ball operates under the same principles as a mouse, the mouse version of the *FlyFitter* can also be used with a trak-ball.

It was decided that use of a touch sensitive screen or a touch tablet could only be made effective using the third technique of parameter variation mentioned above and that their use would not offer significant advantages over the mouse or trak-ball. A keyboard version of *FlyFitter* was discarded early in the project because it was found to be difficult to use in a real-time fitting situation, and it offered no advantages over the joystick and mouse versions of the program.

The Joystick Version

The joystick version of *FlyFitter* was written first because a pair of joysticks was readily available and because it had been originally envisioned that a pair of biaxial joysticks would be the optimum interface for the program. The goal of the joystick approach to phenomenological fitting is to read the joysticks and update the model curve frequently enough to avoid screen flicker, and provide instantaneous feedback to the modeler about the effects of parameter manipulations. In order to achieve this goal, the following things must be done in a cycle at least 8-12 times every second:

1. Erase the old parameter values
2. Read all four joystick axis values
3. Scale and quantize the joystick readings
4. Modify the associated parameters by the appropriate amounts
5. Display the new parameter values

6. Erase the old model curve
7. Recompute the model curve with the updated parameters
8. Plot the new model curve
9. Erase the old value for χ^2
10. Compute a new value for χ^2
11. Display the new value for χ^2

Steps 3, 4, 7, and 10 are very computationally intensive tasks. The need for computational speed required the use of a numeric co-processor in the computer. Although this requirement conflicts with the objective of making the program useable on as many machines as possible, it was realized that the program would never run quickly enough without a fast numeric co-processor.

The joysticks can be read in several ways, but, since speed was so critical in the joystick version of *FlyFitter,* assembly language routines were written to read them. The first routines written used BIOS interrupts to read the joysticks [DUN88], but pure assembly language was found to be quicker [MCA85 & HER89]. The assembly language routines used to read the joysticks are in Appendix A.

A number of serious problems were encountered with the joystick version of *FlyFitter*. One of the most persistent problems was with reading the joysticks too often. Reading a joystick involves charging a capacitor on the joystick card, then timing how long it takes for the voltage across the capacitor to drop to a certain level. The length of time it takes is proportional to the deflection of the joystick that is connected to a variable resistor which is connected across the capacitor. If the joystick is read too often, the capacitor will not fully discharge between readings and erroneous values will result. It is possible to increase the speed with which a joystick may be read by physically modifying the card with new capacitors or by putting additional resistors across the capacitors [HER89]. However, since the objective of the project is to make the program useable on standard equipment, the only solution to the problem was to have the computer perform the other tasks in the cycle between the joystick readings. This requirement resulted in a reduction in the speed of the loop and an overall reduction in the frequency of screen updates.

In order to have a neutral position for the stick when it is centered and a rate of parameter update that is proportional to stick deflection, the binary counts from the assembly language timer described above had to be quantized and scaled into a useable set of multiplicative factors for modifying the parameters. Furthermore, the scaling cannot be done by a constant factor, nor can the quantizing be done in predetermined step sizes because each joystick card returns completely different count values because of differences in the analog components used on them. The counts returned from a fully deflected joystick by the joystick reading routines in Appendix A ranged from the low hundreds with some cards to the high thousands with others. Scaling factors and quantizing step sizes had to be computed by the program and stored. The result of these problems was a very time consuming joystick handling process. This, combined with the speed inherently required by the joystick technique, made the program too slow to be practical except on the fastest of microcomputers.

Another drawback to the joystick technique was that the requirement for continuous updates of the screen resulted in a flickering of the model curve and the parameter values. Although the goal was 8-12 screen updates per second, only about 4 updates were possible using an 8 MHz 80286 based microcomputer with an 80287 math co-processor.

While it was originally envisioned that using a joystick would feel natural to a modeler, several situations arose during testing that clearly were not well suited for the joystick technique. One of them occurred when fitting a damped sinusoidal model like the one described in the introduction to this paper. The parameter ω which represents frequency of oscillation was assigned to the horizontal axis of the joystick. The instinctive thing to do in a situation where it is desired to move the model plot to the right is to move the joystick to the right. However, moving the joystick to the right increased ω which compressed the plot of the model making it appear to move to the left. In this case and many others, the modeler had to fight the instinctive reaction in order to achieve the desired results. This went against the objectives of the design.

Additional problems with the joystick included a lack of common availability and a lack of precision. Needless to say, the joystick version of *FlyFitter* was dropped in favor of the mouse version.

The Mouse Version

The mouse version of the *FlyFitter* was written to operate using a very different type of interface than the joystick version. The modeler specifies an increment value for each of the model parameters before beginning the fitting process. The mouse controls an on-screen pointer that is used to select and press graphical "buttons" on the display. Each of the model parameters are displayed on the screen with their current values and an associated pair of graphical buttons. There are two buttons, one labeled "+", and one labeled "–". These buttons are selected with the mouse pointer and used to increment and decrement the parameter respectively. It was found after some experimentation that this approach offers several distinct advantages over the joystick approach.

Since parameters can only be varied one at a time using the mouse approach, the computational load of the computer is cut by almost a factor of four over the joystick approach. Furthermore, the computer need not update the values nor the display of the values of any of the parameters that are not being modified. Since parameter modification takes place only when a mouse button has been pressed rather than continuously as with the joysticks, the computer can sit idle between parameter modifications. The advantage is that the screen is not updated unless a parameter is modified so screen flicker is eliminated except during continuous parameter modification. Since screen flicker is less of a consideration than before, the program can run effectively with as few as 2-3 screen updates per second.

Since the mouse method does not rely on a measured movement to compute the magnitude of parameter change, there is no requirement to scale or quantize readings from the mouse. The mouse is also serviced to by a dedicated, fast, software mouse driver allowing any dealings with the mouse to be conducted very quickly through BIOS interrupts from within Turbo Pascal [WEI89].

The result of these differences is a new, less demanding cycle of tasks for the computer to perform for each parameter modification. The new cycle is shown below:

1. Wait for a mouse button to be pressed
2. Erase the value of the modified parameter
3. Modify the parameter by the appropriate amount
4. Display the new parameter value
5. Erase the old model curve
6. Recompute the model curve with the updated parameter
7. Plot the new model curve
8. Erase the old value for χ^2
9. Compute a new value for χ^2
10. Display the new value for χ^2

Under the same conditions, this loop operates more than twice as quickly as the loop in the joystick version of the program. However, the computationally intensive tasks in steps 6 and 9 still necessitated the use of a numeric co-processor.

Coding *FlyFitter*

Once the attributes of the display had been decided on and the cybernetic interface had been chosen and developed, the bulk of the computer code that performs the real work had to be written. When writing any sort of computer code where speed and memory are significant considerations, a balance has to be found between the modularization of the code and the efficiency of the code.

Highly modular code is generally easy to read and modify or update. This makes it convenient for debugging or improving. However, modular code is also generally less efficient than in-line code because it relies on the concept of reusable code. Reusable code is code that is called upon by several different parts of the program to perform tasks. The term procedure is used in Pascal for a reusable code segment. The problem is that calling procedures involves jumping from place to place in the code resulting in a fragmented and slow execution path for the code. Additionally, since procedures must be used effectively by several parts of the program, each of which puts slightly different demands on it, the procedure must be generalized for several similar tasks rather than specialized for a specific task. Specialized code is always quicker than generalized code.

In-line code is quicker than modular code because it is placed in the code where it is needed rather than called upon from somewhere else in the program. This results in a continuous and efficient path of execution. Since it is not used by other parts of the program, in-line code can be highly specialized which usually results in high execution speed. The drawback to using in-line code is that if similar tasks are to be performed at several points within a program, the in-line code must be written in at every place it is needed. This uses more memory than modular code. In-line code is also usually more difficult to read and modify later than modular code.

FlyFitter was written using a balance of in-line and modularized code. Most of the tasks performed by the program do not require speed, so these parts of the program were written using many procedures and procedure calls. Only the main fitting loop of the program that is executed every time a parameter is modified used primarily in-line code.

Turbo Pascal offers a feature called the Turbo Pascal Unit (TPU) to help make programs more modular and readable. Each TPU contains procedures and functions that are related to one another allowing the programmer to organize procedures and functions by the type of task they were written to perform. *FlyFitter* was written using twelve TPU's that are all linked to each other and to the main program called *MFlyFit*. The relationship between the various units is shown in Figure 6. All of the TPU's written in italics are built into Turbo Pascal, and the others were written specifically as part of *FlyFitter*. The TPU's *Fonts, Drivers,* and *GetBGI,* which incorporate many of Turbo Pascal's graphics functions into *FlyFitter*, are modified versions of units included in the *Turbo Pascal Users Manual*. The TPU's *Parser,* which is an equation parser, and *FLib,* which contains many of the built in functions available to the equation parser, are rewritten versions of units originally authored by various faculty members at the United States Naval Academy.

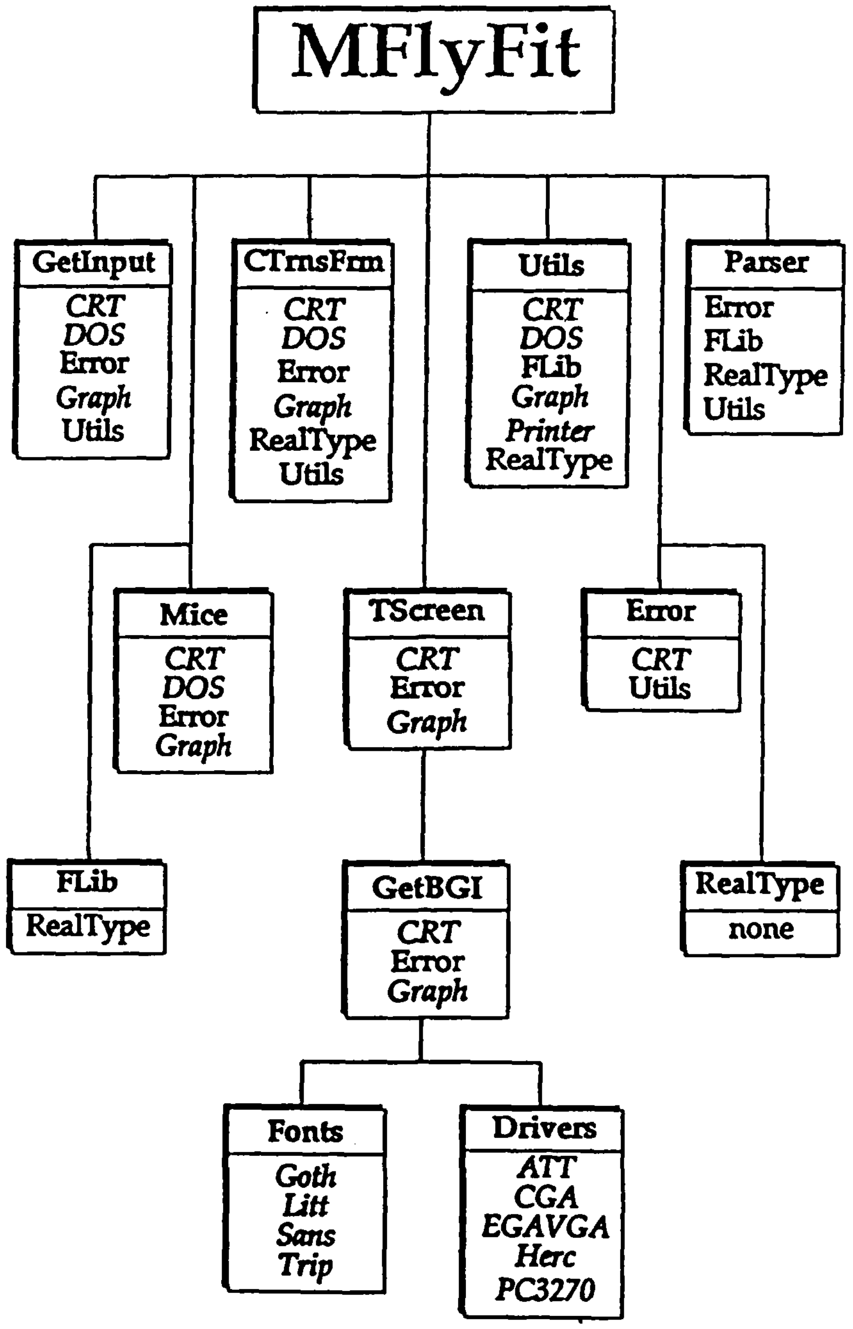


Figure 6: Turbo Pascal Unit (TPU) Organizational Chart

The bulk of the program is contained in *MFlyFit* which contains the code for the main fitting loop and the code for the input screens that accept information about the model and parameters from the modeler. *MFlyFit* relies on *Mice,* which contains mouse handling BIOS interrupt routines, to read the mouse, and it relies on *Ctrnsfrm* to do coordinate transformations between the real world coordinates of the data points and the screen coordinates used by the computer. *GetInput* contains routines that control the input of information by the modeler through the keyboard and screen the information for invalid input. It also includes routines to display files on disk and retrieve them. Finally, *Error* and *Utils* contain procedures and functions that perform a myriad of tasks required by all of the other TPU's.

CASE STUDIES

Once a preliminary version of *FlyFitter* had been completed, the next step in the project was to make it available to professors and midshipmen to use in their research or class work. The aim was to have individuals who were not directly connected with the project test the program using actual data taken in the laboratory and provide comments about its effectiveness. The general response was very positive, but most individuals who used the program had suggestions for improvements. Comments and suggestions resulting from the original distribution of the program gave rise to the following modifications in *FlyFitter*:

1. The maximum number of data points in a data file was increased from 300 to 600.
2. The maximum length of a model function was increased from 80 characters to 240 characters.
3. Spherical Bessel functions were added to the function library.
4. The dimensions of the graphics window on the fitting screen were increased by about 10%.
5. Some of the input screen commands were changed to make commands standard throughout the program.
6. The layouts of the input screens were changed to make them more readable.

Once these modifications had been made, *FlyFitter* was in its final form for the purposes of this project. The final task was to use the phenomenological fitting procedure in some real-world situations. Detailed accounts of two of the instances in which it was effectively used are given [illegible]w.

Desig[illegible] of a Heat Exchanger

Midshipman Rich Wells and Midshipman Doug Reckamp were assigned a design project in which they were to design a compact heat exchanger for use as an intercooler in an LM2500 gas turbine. The gas turbine was to operate with a fixed airflow rate and fixed temperatures and pressures on each side. The coolant to be used was sea water. Midshipmen Wells and Reckamp were given that the frontal dimensions of the heat exchanger must be between 0.5 and 2 meters, that the air-side pressure loss must be less than 2% of the inlet pressure, and that the temperature differential was not to exceed 70 °R. A schematic diagram of the heat exchanger is shown in Figure 7.

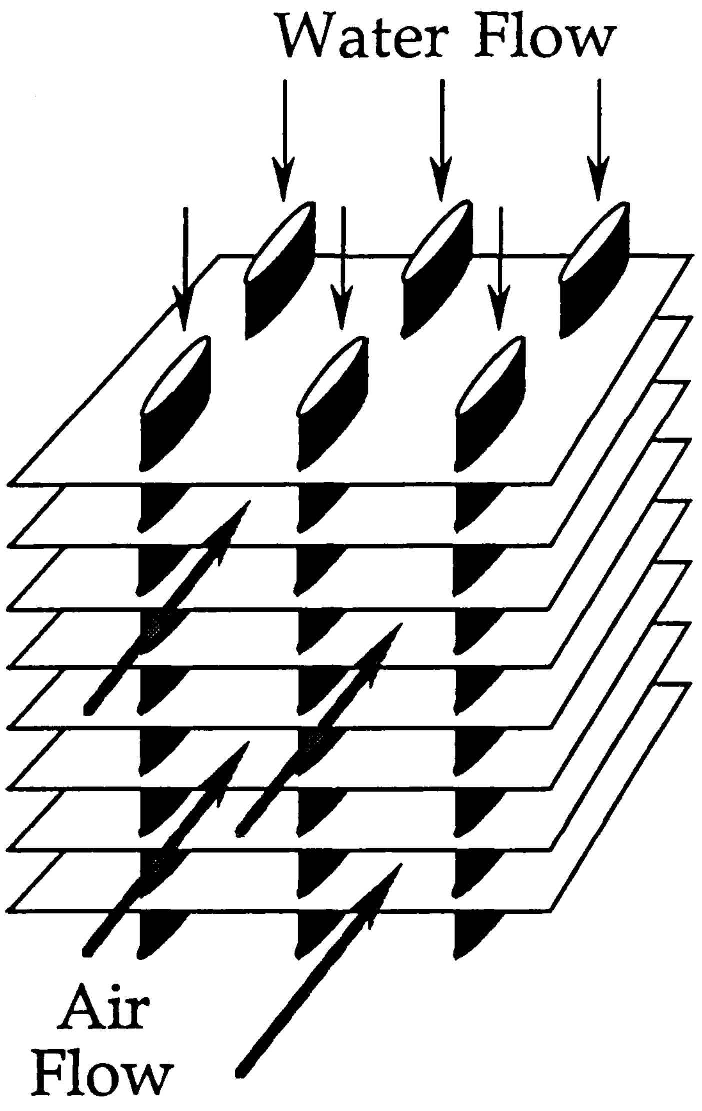


Figure 7: Heat Exchanger

In designing the heat exchanger, two instances were encountered where *FlyFitter* proved useful. The first stemmed from the fact that the convective heat coefficient for air is dependent on the Stanton Number (St). Since only the Reynolds Number (Re) and the Prandtl Number (Pr) were known, a means was needed to determine the Stanton Number given the Reynolds Number and the Prandtl

Number. A graph of St $Pr^{2/3}$ vs. Re was found in a textbook and the values were taken from the graph and made into a data file. The model chosen to fit the data was

$$y_t(x_i) = a + b x_i^{-1} + c x_i^{-2} + d x_i^{-3} \ ,$$

a linear model. Figure 8 shows the results of fitting the model using *FlyFitter v1.0* which is included on disk with a brief users manual in Appendix B.

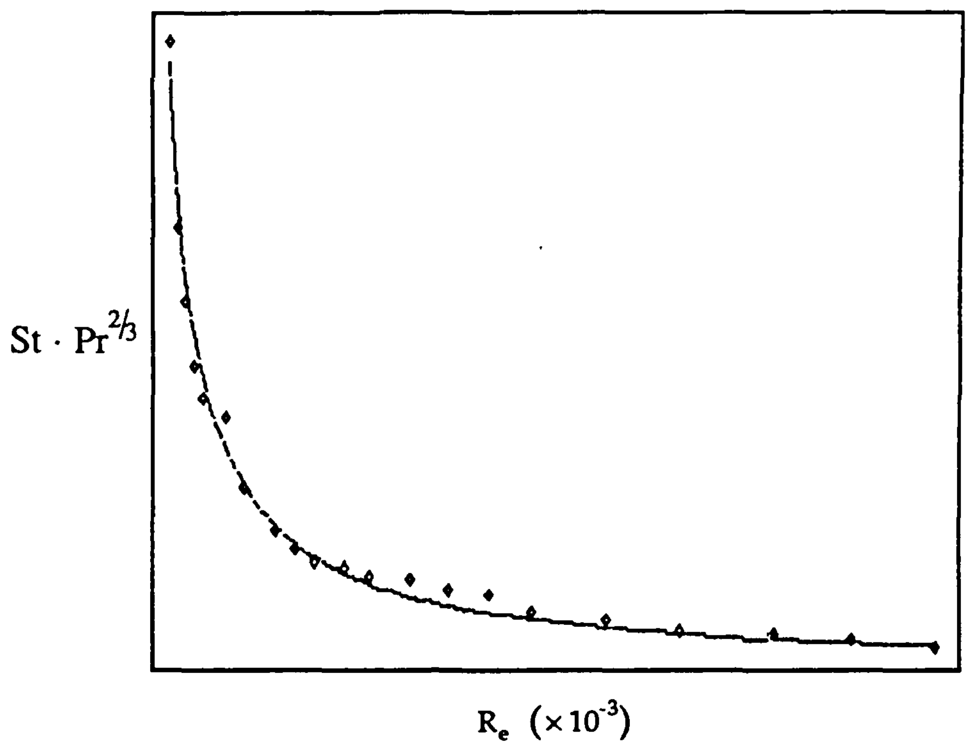
The second time phenomenological fitting proved useful was in determining the pressure drop across the heat exchanger. Again no direct relationship between Reynolds Number and pressure drop was available, but a graph of friction factor (*f*) vs. Re was found in a textbook and the values were taken from the graph and made into a data file. This time, the model selected was

$$y_t(x_i) = a + b x_i^{-1} + c x_i^{-2} \ ,$$

again a linear model. Figure 9 shows the results of the phenomenological fitting.

Once the values of the parameters had been determined using *FlyFitter,* the models were used as relationships in TK-Solver which determined the optimal design for the compact heat exchanger.

(Stanton#) · (Prandtl#)$^{2/3}$ vs. Reynolds Number



```
******************* FlyFitter v1.0 *******************

Model Function:  f(x) = a+b/x+c/(x^2)+d/(x^3)
```

or $f(x) = a + b\,x^{-1} + c\,x^{-2} + d\,x^{-3}$

```
X^2:  0.000000048
```

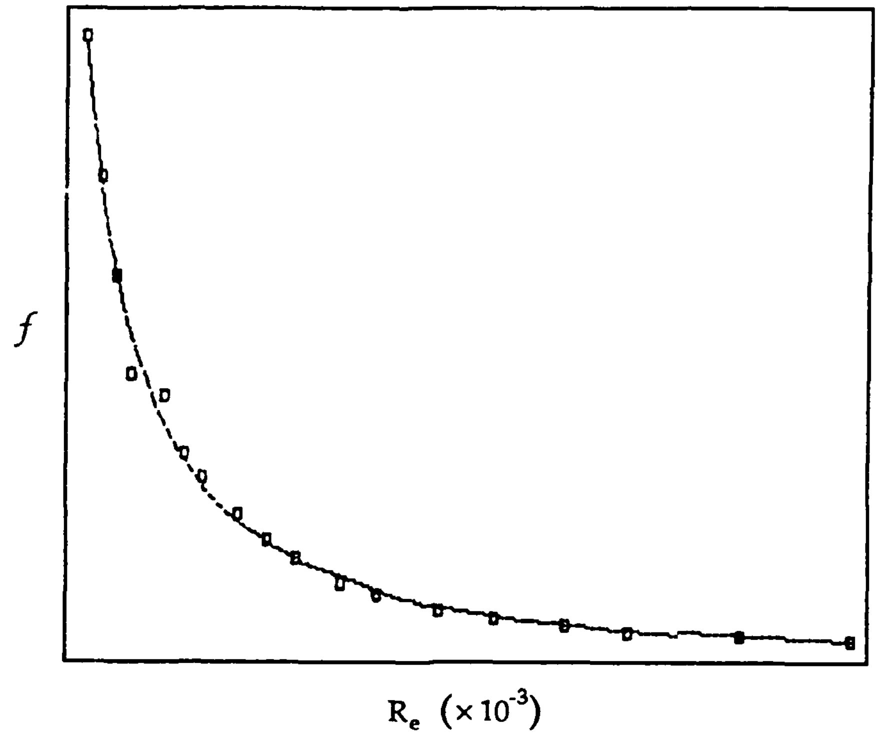
⇐ *this is* χ^2

```
Parameters:

          a  =   0.00367
          b  =   0.00357
          c  =  -0.00042
          d  =   0.000111
```

Figure 8: First Fit for Heat Exchanger

Friction Factor vs. Reynolds Number



```
****************** FlyFitter v1.0 ******************

Model Function:  f(x) = a+b/x+c/(x^2)
```

or $f(x) = a + b\,x^{-1} + c\,x^{-2}$

```
X^2:  0.0000004226
```
$\Leftarrow$ *this is* χ^2

```
Parameters:

          a  =   0.00936
          b  =   0.01334
          c  =   0.00091
```

Figure 9: Second Fit for Heat Exchanger

Non-Linear Scattering of Focused, Crossed, Sound Beams by Turbulence in Water

Associate Professor Murray Korman of the Naval Academy's Physics Department provided data that had been taken in his acoustics laboratory. The data had been taken as part of a Trident Project conducted by Steve Rife during the academic year 1987-88. The apparatus used in the research is shown in Figure 10. Two transmitters operating at slightly different frequencies, emitted sound into the turbulent flow caused by a jet of water flowing in a direction perpendicular to the line between a receiver and the transmitters. The receiver then measured the non-linear scattering intensity of the sound on the opposite side of the flow at various distances from the center of the water jet.

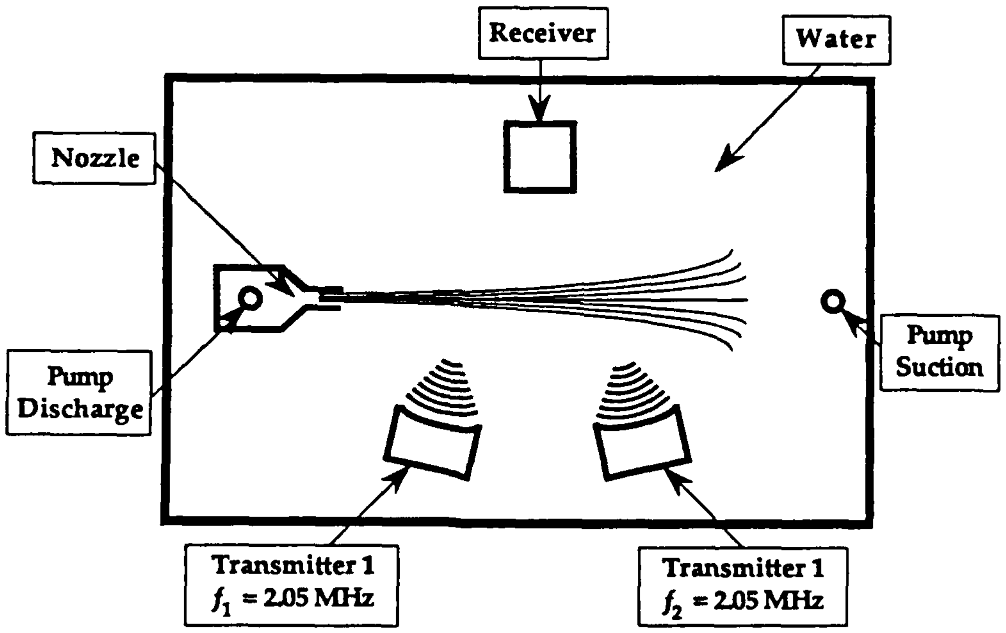

Receiver
Water
Nozzle
Pump Discharge
Pump Suction
Transmitter 1 f_1 = 2.05 MHz
Transmitter 1 f_2 = 2.05 MHz

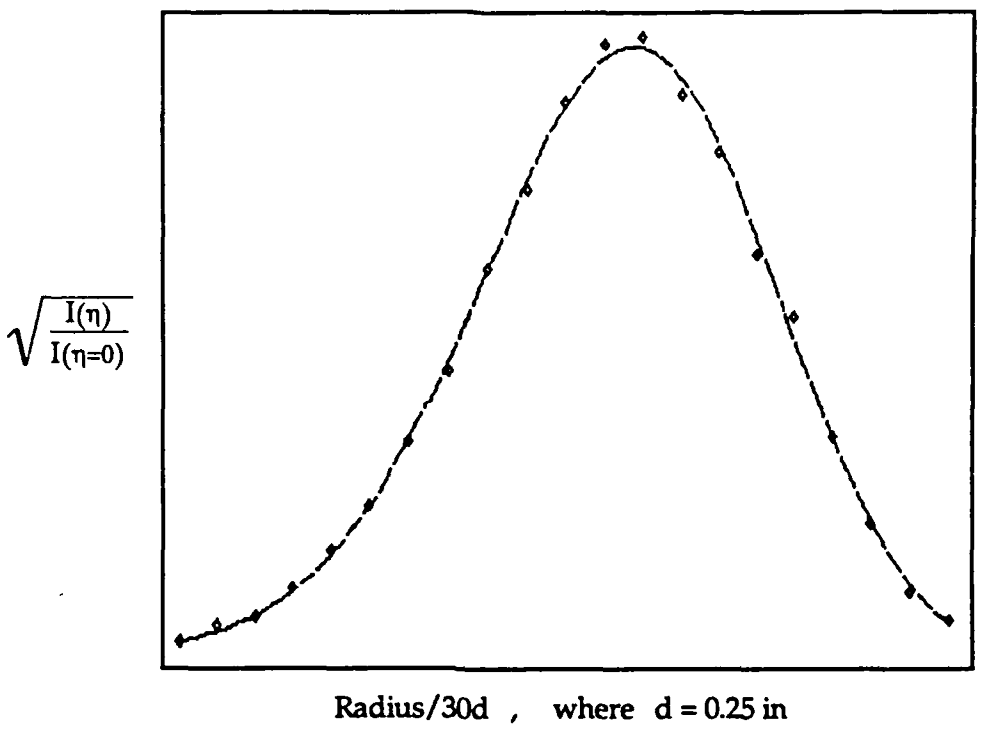
Figure 10: Apparatus for the Analysis of Non-Linear Scattering of Focused, Crossed, Sound Beams by Turbulence in Water

The data were collected in pairs of non-linear scattering intensity vs. radius, and the elements of the data pairs were converted into dimensionless ratios before fitting began. The non-linear model selected by Professor Korman was

$$y_t(x_i) = \left(b + c\,x_i + d\,x_i^2 + f x_i^3\right) e^{-a\,x_i^2} \quad .$$

Figures 11 and 12 show the results of two fits of the model to different sets of data.

Non-linear Scattering Intensity vs. Scaled Radius



```
******************** FlyFitter v1.0 ********************

Model Function:  f(x) = (b+c*x+d*x^2+f*x^3)*exp(-a*x^2)
```

or $f(x) = \left(b + c\,x + d\,x^2 + f\,x^3\right) e^{\left(-a\,x^2\right)}$

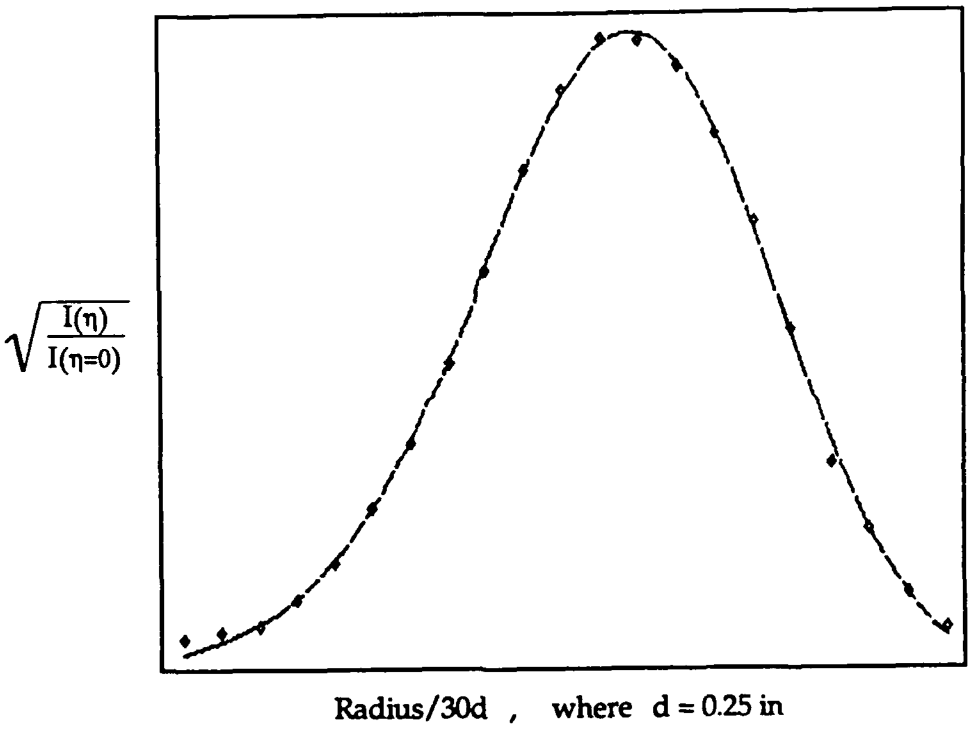
```
X^2:  0.000225245
```
$\Leftarrow$ *this is* χ^2

```
Parameters:

          a  =  51.266842
          b  =   0.980012
          c  =   0.56140477
          d  =  -9.5811328
          f  = -42.153919
```

Figure 11: First Fit for Non-Linear Scattering

Non-linear Scattering Intensity vs. Scaled Radius



```
******************* FlyFitter v1.0 *******************

Model Function:  f(x) = (b+c*x+d*x^2+f*x^3)*exp(-a*x^2)
```

or $f(x) = \left(b + c\,x + d\,x^2 + f\,x^3\right) e^{\left(-a\,x^2\right)}$

```
X^2:  0.0001385925
```

⇐ *this is* χ^2

```
Parameters:

          a  =  53.646342
          b  =   1.007312
          c  =   0.68350477
          d  =  -7.5654328
          f  = -54.396719
```

Figure 12: Second Fit for Non-Linear Scattering

PHENOMENOLOGICAL FITTING TODAY AND TOMORROW

Phenomenological fitting is still in its infancy as a serious fitting tool, but the potential it offers will make it part of fitting in the future. Already it brings the power to fit complex non-linear models to anyone with access to a properly equipped microcomputer. For those with access to faster computers and rigorous fitting algorithms, phenomenological fitting is a means by which parameter seeds may easily be determined and used to save time and money on main-frame and mini-computers.

As an educational tool, phenomenological fitting can help the modeler to better understand the sensitivity of the model to variations in model parameters. Though it was designed to tackle complex non-linear model fits, phenomenological fitting techniques can just as easily handle the simple linear fits encountered by high school and undergraduate college students while enabling them to understand the fitting process more completely than with conventional fitting algorithms.

In the future, phenomenological fitting will enjoy the benefits of faster computer hardware and higher resolution graphics. An increase in speed will make fitting more complex models practical and increased display resolutions will mean greater precision can be achieved using visual fitting techniques. One of the most serious restrictions of phenomenological fitting is that it is restricted by its nature to fits involving only one independent variable. As holographic technology advances, however, it may become possible to display true three dimensional graphics, making fits involving two independent variables possible.

REFERENCES

[BEV69] Philip R. Bevington, *Data Reduction and Error Analysis for the Physical Sciences*, McGraw-Hill, 1969.

[DAN80] Cuthbert Daniel and Fred S. Wood, *Fitting Equations to Data; Computer Analysis of Multifactor Data*, John Wiley & Sons, 1980.

[DUN88] Ray Duncan, *Advanced MS-DOS Programming*, 2nd ed., Microsoft Press, 1988.

[ERT88] John P. Ertel, "The Phenomenological Fitting Procedure and the Fast Microcomputer," Proceedings of the *4th International Conference on Systems Research, Informantics, and Cybernetics*, Baden-Baden, West Germany, 1988.

[ERT87] John P. Ertel, "Fly Your Own Fit (or Game Graphics Techniques Come to Modeling)," Proceedings of the *18th Annual Pittsburgh Conference on Modeling and Simulation*, v18/5 p1719: Robots and General Modeling, Session 6-7, 1987.

[HER89] Allan Herr, Telephone Interview, September, 1989.

[LIC88] William Lichten, *Data and Error Analysis in the Introductory Physics Laboratory*, Allyn and Bacon, 1988.

[MCA85] James P. McAdams, "IBM PC Joystick Control Using Turbo Pascal," *BYTE*, October, 1985: 143-144.

[SUT88] George Sutty and Steve Blair, *Programmer's Guide to the EGA/VGA*, New York: Brady, 1988.

[TAY82] John R. Taylor, *An Introduction to Error Analysis; The Study of Uncertainties in Physical Measurements*, University Science Books, 1982.

[WEI89] Keith Weiskamp, Loren Heiny, and Namir Shammas, *Power Graphics Using Turbo Pascal*, Wiley, 1989.

APPENDIX A

The following three assembly language subroutines calibrate and read joystick deflections. The AH register must contain a mask value (01 - X-axis of joystick A, 02 - Y-axis of joystick A, 04 - X-axis of joystick B, 08 - Y-axis of joystick B) before *READ_AXIS* is called. The routine *CALIBRATE* is called only when it is desired to recalibrate the joysticks.

```
                MOV       AH,AXIS_MASK
READ_AXIS       MOV       DX,0201
                MOV       CX,0400
L1:             IN        AL,DX
                TEST      AH,AL
                LOOPNZ    L1
                MOV       CX,0000
                CLI
                OUT       DX,AL
                JMP       L2
                NOP
                NOP
L2:             IN        AL,DX
                TEST      AH,AL
                LOOPNZ    L2
                STI
                NEG       CX
                RET

CALIBRATE       MOV       AH,01
                CALL      READ_AXIS
                SHL       CX,1
                MOV       AX_SCALE,CX
                RET

GET_VALUE       MOV       AH,01
                CALL      READ_AXIS
                XOR       DX,DX
                MOV       AX,CX
                SHL       AX,1
                SHL       AX,1
                SHL       AX,1
                SHL       AX,1
                SHL       AX,1
                MOV       CX,AX_SCALE
                IDIV      CX
                RET
```

APPENDIX B

Instructions for Using *FlyFitter v1.0*

A copy of *FlyFitter v1.0* has been included as an executable file on disk for use in any scientific or educational applications required by the reader. Below are complete instructions designed to help the reader use the program effectively.

System Requirements

512K RAM
1 floppy or fixed disk drive
16 color EGA graphics card with color EGA screen
Logitech compatible mouse
8087, 80287, or 80387 math co-processor

Loading & Running

The program may be run from the distribution disk. The disk is bootable and will load MS-DOS and the Logitech mouse driver then automatically load and run the program. To do this, put the distribution disk in drive A: and turn on the machine. If all is well, the title screen will appear. If the title screen fails to appear and an error message is displayed in the upper left-hand quarter of the screen, check the computer hardware; the program will report an error if the computer lacks the necessary hardware or is improperly set-up.

Most mice should be compatible with the program so long as they have an appropriate software driver. However, the program was developed for the Logitech mouse and works best with this type of mouse. If a mouse other than a Logitech mouse is used, MS-DOS and the appropriate mouse driver must be loaded separately before beginning the program. The program can then be run from the disk by inserting the disk in to drive A, typing MFLYFIT at the A> prompt and pressing RETURN.

If the program is to be run from a fixed disk, create a directory on the disk and copy MFLYFIT.EXE from the distribution disk to the directory. Several example data files and fit parameter files have been included on the disk as examples to aid in the learning process. These example files may be included in the fixed disk directory by copying the files with .DAT and .FPM extensions to the directory. The program may then be run by typing **MFLYFIT** at the C> prompt from within the directory that contains the program.

To optimize the program's performance, do not use memory resident software with it, and use the fastest computer available. For complex fits, an 80386 running at 25 MHz may be necessary to achieve satisfactory results. Even for simple fits, it is recommended that an 80286 running at 8 MHz or higher be used.

Considerations When Using *FlyFitter*

The program must be provided with the following information before it can be used:

1) A set of data in text file format
2) A mathematical model with no more that 16 parameters that is to be fit tc the data
3) Single letter designations for each of the parameters in the mathematical model
4) Initial values for each of the parameters
5) An initial rate of change for each of the parameters that will determine how quickly each parameter is varied during fitting
6) If a printed report is required, be certain that an appropriate printer is on-line

The following things should be considered when choosing these options:

1) The program frequently computes a value for χ^2, so the fewer the data points in the data file, the faster the program will run. In its current form, the program will handle up to 600 data points in a data file. If a file with more than 600 data points is used by the program, all points beyond number 600 will be ignored.

2) The plot resolution should be kept as low as possible, again to speed up the program. This option may be chosen on the Main Input Screen (MIS). When this value is chosen, it should be kept in mind that higher resolutions may be necessary to avoid confusing aliasing for mathematical models that oscillate.

3) Make certain that every parameter that appears in the mathematical model also uniquely appears on the Parameter Update Screen

(PUS). For example, if z is to be used as a parameter, make certain that it appears uniquely on the PUS. All lowercase letters except x and e are legal parameter designations. e is reserved for the constant 2.71... and x must be the independent variable in the mathematical model.

4) The program allows up to six parameters to be designated "active" at any given time. Only active parameters may be varied at fit time. If it is desired to vary an inactive parameter, it must be made active first by placing a "Y" in the active column of the PUS.

Creating a Data File

FlyFitter must be provided with a data file in order to function. The data file should be a text type file with proper delimiters between each entry (usually a [tab]). The format should be x y x y x y x .... A carriage return may be placed after each data pair, but it is not necessary to do so. An example of a valid file is below:

23.45 [tab here] 342.43 [carriage return here]
34.553 [tab here] 534.32 [carriage return here]
45.901 [tab here] 693.33 [optional carriage return here]

Do not put extraneous characters or unnecessary carriage returns in the file. Most word processors and many spreadsheets are capable of creating this type of file automatically. When using a word processor, save the file as an ASCII file or a text file. If further guidance is required, please look at the files with a .DAT extension included on the distribution disk; these are all valid files.

Using *FlyFitter*

After the program has loaded and run, the title screen should be displayed. Press **RETURN** to get past the title screen, and the Main Input Screen should appear. From here, the blanks on the screen may be filled out, or one of the options displayed at the bottom of the screen may be exercised. To move between fields on any input screen, use the **UP-ARROW** and **DOWN-ARROW** keys (be sure the **NUM-LOCK** key is off). To move within a field, use the **LEFT-ARROW** and **RIGHT-ARROW** keys. The **HOME** key moves the cursor to the beginning of a field and the **END** key moves the cursor to the end of a field. The **BACKSPACE** key erases the character to the left of the cursor and moves the cursor left one space. The **DELETE** key erases a character to the right of the cursor. The INSERT key toggles between insert mode and type-over mode. Pressing ALT-B will clear the current field. Only legal characters may be entered into the fields, so if the program ignores a key-press, it is probably because an attempt was made to enter an illegal character.

The fields on the MIS include the type of plotting symbol to be used for the data, the mathematical model, and the resolution to be used for the plot generated by the mathematical model.

When entering the mathematical model, be aware that a wide variety of functions are available. Below is a list of the functions that are supported in addition to the usual array of functions, and their correct entry format (a, x, and y imply real numbers, n and m imply integers):

angle(x,y)	arccos(x)
arcsin(x)	arcsec(x)
cosh(x)	cot(x)
sinh(x)	tanh(x)
coth(x)	csc(x)
csch(x)	factorial(n)
inf(x,y)	intpower(x,n)
invcosh(x)	incsinh(x)
invtanh(x)	invcoth(x)
invsech(x)	invcsch(x)
loga(a,x)	power(x,a)
ratpower(x,n,m)	sec(x)
sech(x)	sup(x,y)
tan(x)	heav(x)
log10(x)	exp10(x)
sround(x)	strunc(x)
sgn(x)	sbj(n,x)

The program will not let the cursor leave the equation field until a valid mathematical model has been entered. The default model is x. The letter x must be used as the independent variable in the model.

All options at the bottom of each screen are selected by pressing and holding the ALT key while simultaneously pressing the character of the desired option.

Before the Fit option **ALT-F** may be selected, a data file must first have been loaded using the **ALT-D** option. Then, the computer must be provided with the mathematical model and the necessary information about its parameters. The model must be entered on the MIS, and all of the parameter information must be entered on the PUS which can be accessed by selecting the **ALT-U** option. To return to the MIS from the PUS, select the **ALT-P** option which will almost always cause the computer to return to the previous screen from wherever it is. Once the information about the model has been entered once, it need not be done again. All the information may be stored in a file by selecting the **ALT-S** option and later recalled using the **ALT-R** option. A filename must be provided when the **ALT-S** option is selected. An extension of .FPM is automatically appended to any filename provided, so it is not necessary to include a filename extension.

The **ALT-R** and **ALT-D** options first ask for a proper path and series of wildcards which describe the set of files from which the user wishes to select. Wildcards and paths are explained thoroughly in the MS-DOS manual. If it is unclear how to designate a proper path and set of wildcards, just accept the default path and wildcards whenever this screen is presented. After selecting a path, the computer will display all of the files that meet the specified path and wildcard restrictions. The arrow keys may be used to move among these, and the **RETURN** or **ENTER** key selects and loads the currently highlighted file. If there are more files than may fit on the screen, then use the **PAGE-UP** and **PAGE-DOWN** keys to display the other files. The **HOME** and **END** keys move the highlight block to the first and last entries on a screen respectively.

The **ALT-G** option prints a simple report on the printer. The first page includes the mathematical model, the most recent value for χ^2, and the fit parameters. To include data in the report, simply respond by pressing Y to the inquiry. The data will be printed out on the second and subsequent pages in a tabular format.

Finally, the **ALT-F** option begins the fitting process. If the operator has loaded a valid data file and has properly completed the MIS and PUS, selecting this option should result in a graph on the upper right of the screen that includes the data plotted with the designated symbol, and a green curve that represents the mathematical model computed with the current parameters. Each of the active parameters and their values along with χ^2 and its value should be displayed on the far left. At the bottom should be each of the active parameters with a + and – button on either side. The mouse cursor is active and ready to use. To vary a parameter, simply move the mouse cursor to the appropriate button and press the mouse button. Holding down the left mouse button will result in repeated variation of the parameter. The parameter will be varied by the incremental value designated on the PUS. Pressing the middle and right buttons will result in varying the parameter by 3 and 10 times this increment respectively. To increment the parameter, use the + button to the right of the parameter and to decrement the parameter, use the – button to the left of the parameter. The χ^2 value and the green curve will be updated appropriately every time you vary a parameter. The best technique for obtaining a fit is to vary parameters until the green curve matches the grey data points visually. Then, fine adjustments may be made by attempting to minimize the value for χ^2 which is displayed on the lower left of the screen. Very high precision is possible by continually decreasing the incremental values for the parameters. A small amount of practice will result in operator skill in parameter adjustment. A detailed understanding of the mathematical behavior of the model is also helpful but not required.

At the bottom right corner of the screen, there is a button labeled **8** and a button labeled **24**. These buttons will produce graphics dumps of the plot area of the screen on Epson compatible 8 or 24 pin printers. To avoid "hanging" the computer, be certain that a compatible printer is properly connected and set up before selecting one of these options.

The **REDRAW** button exists because clutter will inevitably appear on the screen due to unavoidable rounding errors. If the clutter becomes distracting, simply select the **REDRAW** button and the clutter should disappear. To return to the MIS, select the **EXIT** button.

Several example data and parameter files have been included on the disk to help the reader learn to use . Each of the examples includes two files; one with a .DAT extension, and one with a .FPM extension. The files can be loaded from within *FlyFitter* and used to practice curve fitting.

Disk (FlyFitter v1.0, rev.2) located in pocket of original manuscript (see Nimitz Library, Special Collections).

MFLYFIT